Declarative Process Mining for Software Processes: The RuM Toolkit and the Declare4Py Python Library

Anti Alman[1(✉)], Ivan Donadello[2(✉)], Fabrizio Maria Maggi[2(✉)], and Marco Montali[2(✉)]

[1] University of Tartu, Tartu, Estonia
anti.alman@ut.ee
[2] Free University of Bozen-Bolzano, Bolzano, Italy
ivan.donadello@unibz.it, {maggi,montali}@inf.unibz.it

Abstract. Process mining is one of the research disciplines belonging to the field of Business Process Management (BPM). The central idea of process mining is to use real process execution logs in order to discover, model, and improve business processes. There are multiple approaches to modeling processes with the most prevalent one being the procedural models like Petri nets and BPMN models. However, procedural models can be difficult to use for processes like software processes that are highly variable and can have a high number of different branches and exceptions. In these cases, it may be better to use declarative models, because declarative models do not aim to model the end-to-end process step by step, but they constrain the behavior of the process using rules thus allowing for more flexibility in the process executions. The goal of this paper is to introduce the main principles of declarative process mining (i.e., process mining based on declarative models) and to show which state-of-the-art declarative process mining techniques have been implemented in the RuM toolkit and in the Declare4Py Python library.

Keywords: Declarative Process Mining · Software Processes · Process Discovery · Conformance Checking · Process Modeling

1 Introduction

Business Process Management (BPM) has become an integral part of how companies organize their workflows starting from the higher levels of management as recommended in ISO 9000, ISO 9001 Quality Management Principles (especially principles 4, 5, and 6)[1] to modeling and optimizing lower level processes through the use of various process mining techniques [1]. Process mining is the part of BPM which is focused on the analysis of business processes based on event

[1] International Organization for Standardization, *ISO quality management principles*, 2015: https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100080.pdf.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
R. Kadgien et al. (Eds.): PROFES 2023, LNCS 14484, pp. 13–19, 2024.
https://doi.org/10.1007/978-3-031-49269-3_2

logs containing information about process executions. Process mining techniques are usually divided into three main branches which are process discovery, conformance checking, and process enhancement. Process discovery is used to generate a model of the process based on an event log of the process. Conformance checking is used to compare an event log to a process model with the aim of finding discrepancies between the event log and the model. Process enhancement is used to modify the model based on the information retrieved from the event log.

Process models can be divided into two types. The most common type includes procedural models like Petri nets and BPMN models, which aim to describe end-to-end processes and allow only for activities that are explicitly triggered in the control-flow [4]. However, modeling step by step the entire control-flow can be undesirable in some cases. For example, for software processes that are less structured and can have a high number of different branches and exceptions, the model could become quickly unreadable. For these processes, it may be a better choice to use declarative process models that model the process as a set of constraints that the process should follow. Declarative process mining techniques are process mining techniques based on declarative process models.

This paper presents two tools RuM [3] and Declare4Py [13] that make some of the existing declarative process mining techniques more accessible to a wider range of specialists, by focusing, in particular, on the declarative modeling language Declare [21]. RuM[2] is a Java application providing an easy-to-use GUI. Declare4Py[3] is a Python API that can be easily integrated in other Python applications. The two tools cover similar declarative process mining functionalities with minor differences.

2 Background

This section provides the necessary background on concepts that are crucial for understanding the research topic of this paper. It includes an overview of the process mining artifacts and covers the basics of Declare.

2.1 Event Logs, Traces, and Events

Process mining uses data collected from the information systems that leave their footprint during the executions of the processes they support in the so-called event logs. An *event log* is a set of traces. A *trace* is an execution of a business process. A trace contains a sequence of events, where each *event* is related to the execution of an *activity*, performed at a certain *timestamp* with a (possible) set of other attributes a.k.a. the *payload* of the event. Two traces belong to the same *variant*, if the sequence of activities corresponding to the events in the two traces are the same.

[2] https://rulemining.org.
[3] https://declare4py.readthedocs.io/en/.

2.2 Declare

DECLARE is a modeling language that uses a constraint-based declarative approach to define a loosely-structured process model [21]. The language is grounded in Linear Temporal Logic for finite traces (LTL_f) [10], but no knowledge of temporal logics is required to use the language effectively. The aim of the language is to describe a process in such a way that all the important aspects of the process are defined (such as activity A occurs immediately after activity B), while not requiring the entire process with all of its details to be modeled.

The main building blocks of the language are constraints. Each constraint consists of a template (a constraint type) and a reference to one or two activities depending on the template. A template basically defines the semantic meaning of the constraint and activities in the constraint define the activities to which this meaning applies. For example, if the constraint template is EXACTLY1 and the activity is A then this means that activity A should be performed exactly once during a single process execution.

When working with DECLARE constraints, it is important to understand three main concepts, which are constraint *activation*, constraint *fulfillment*, and constraint *violation*. For each constraint, there is at least one activity that is considered the activation of the constraint. If the activation occurs during the process execution then the corresponding constraint is considered to be activated. The occurrence of an activation triggers some obligations on the occurrence of another activity (the target). For example, for the RESPONSE constraint having A as activation and B as target, the execution of A forces B to be executed eventually after A. When a constraint is activated then it must be either fulfilled or violated by the end of the process execution. An activated constraint will be fulfilled when the condition defined by the constraint is satisfied, otherwise the constraint will be violated. If a constraint is not activated during the process execution then the constraint is considered to be vacuously satisfied [22].

MP-DECLARE [6] is an extension of DECLARE allowing the modeler to specify data conditions on the payload of the activation and/or of the target of a constraint.

3 RuM and Declare4Py

In this section, we list the declarative process mining techniques implemented in the RUM toolkit and the DECLARE4PY Python library. Both tools rely on well-known standards for the input and the output files, i.e., XES [14] for event logs and `decl` [23] for DECLARE models. This ensures their interoperability with other libraries and tools.

3.1 Features of the Tools

RUM currently has five major features: *process discovery*, *conformance checking*, *process monitoring*, *log generation*, and *log filtering*. RUM also provides a DECLARE editor. DECLARE4PY has five major features: *process discovery*, *conformance checking*, *query checking*, *log generation*, and *log filtering*.

Process Discovery. RuM implements four methods for process discovery: Declare Miner, MINERful, MP-Declare Miner, and MP-MINERful. Declare Miner [16] and MINERful [12] are well established discovery algorithms for Declare. The other two algorithms [15,17] are focused on MP-Declare and have an additional post-processing step to discover data conditions from the event log. The result of the discovery task are presented in three different formats: as a Declare model, as a textual description of the discovered constraints, and as an automaton representing the translation into an automaton of the LTL_f semantics of the discovered model. The discovery method supported by Declare4Py is the (data-agnostic) Declare Miner only. The results are returned in a Python data structure containing, for each constraint of the discovered model, the traces that satisfy it. A Declare4Py function allows the user to filter such data structure to retrieve the most frequently satisfied constraints.

Conformance Checking. In RuM, conformance checking is supported by three methods: Declare Analyzer, Declare Replayer, and Data-Aware Declare Replayer. The Declare Analyzer, introduced in [6], takes a model and an event log as inputs and returns activations, violations, and fulfillments in each trace in the log of each constraint in the model. The Declare Replayer [9] and the Data-Aware Declare Replayer [5] report trace alignments (using as inputs models without and with the data perspective, respectively). The results are grouped by trace or by constraint. If the results are grouped by trace, the details of a group show how the selected trace is affected by each constraint in the model. If the results are grouped by constraint, the details of a group show how that specific constraint affects each trace in the event log. The conformance checking method supported by Declare4Py is the Declare Analyzer. The results are listed in a Python data structure indexed by trace identifier. The user can easily query such data structure to retrieve or aggregate information.

Query Checking. This task follows the method presented in [18] and takes as input an event log, a support threshold, and an MP-Declare query (i.e., an MP-Declare constraint in which the activation and/or the target activity are unspecified and replaced with placeholders) and returns the set of assignments of activities to the placeholders such that the input query instantiated using those assignments is satisfied in a percentage of traces in the log higher than or equal to the support threshold. This task is only implemented in Declare4Py and returns a data structure containing the assignments.

Process Monitoring. RuM also provides a functionality for process monitoring. It allows users to animate a Declare model by replaying a log over the model and showing if each trace in the log violates or satisfies the constraints defined in the model while the trace develops. This allows users to simulate the log behavior over a Declare model, observe and identify where the constraints are temporarily or permanently satisfied or violated after the occurrence of each

event in the log. RuM supports three monitoring methods: MP-Declare w Alloy [19], MobuconLTL [20], and MobuconLDL [8].

Log Generation. Generating an artificial event log from a model can be useful for testing out new process mining algorithms or gaining a better understanding of how the process executions may look like based on the model. In RuM, the log generation supports the Alloy Log Generator [18], the ASP Log Generator [7], and the MINERful Log Generator [11], with the main difference being that the Alloy Log Generator and the ASP Log Generator methods can also account for the data perspective in the input model. Also Declare4Py provides a log generator based on ASP. The log generator available in Declare4Py has more advanced options with respect to the ones available in RuM. In particular, the tool is able to generate compliant and non-compliant (positive and negative) traces. The tool is also able to generate multiple variants of compliant and non-compliant traces. The user can specify how many variants should be generated and how many times the same variant should be repeated in the generated log. The tool also allows users to specify the number of times an activation of a constraint must occur in a trace.

Log Filtering. RuM and Declare4Py both provide features for log filtering. As basic filters the ones implemented in Disco[4] are available. A number of advanced filters based on LTL_f are also available. Some of them filter out traces that are not compliant with some predefined LTL_f rules that according to the literature [2] are relevant in the BPM context. Similarly, Declare constraints and Branched Declare[5] constraints can also be used for filtering.

4 Summary

In this paper, we presented the functionalities provided by the RuM toolkit and the Declare4Py Python library. These tools will be presented in a tutorial at PROFES 2023. The tutorial will provide a general overview of the main process mining tasks followed by a brief introduction to the existing declarative process mining techniques. In order to develop a mastery of these techniques, in the tutorial, we will introduce the tools presented in this paper and we will apply these tools to answer process-related questions using real-life datasets pertaining to software processes.

References

1. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19

[4] https://fluxicon.com.

[5] Branched Declare is an extension of Declare in which constraints are defined over disjunctions of activities.

2. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process mining and verification of properties: an approach based on temporal logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005). https://doi.org/10.1007/11575771_11
3. Alman, A., Di Ciccio, C., Haas, D., Maggi, F.M., Nolte, A.: Rule mining with RuM. In: ICPM, pp. 121–128 (2020)
4. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
5. Bergami, G., Maggi, F.M., Marrella, A., Montali, M.: Aligning data-aware declarative process models and event logs. In: BPM, vol. 12875, pp. 235–251 (2021)
6. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
7. Chiariello, F., Maggi, F.M., Patrizi, F.: ASP-based declarative process mining. In: AAAI, pp. 5539–5547. AAAI Press (2022)
8. De Giacomo, G., De Masellis, R., Maggi, F.M., Montali, M.: Monitoring constraints and metaconstraints with temporal logics on finite traces. ACM Trans. Softw. Eng. Methodol. **31**(4), 68:1–68:44 (2022)
9. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the disruptive effectiveness of automated planning for LTLf-based trace alignment. In: AAAI, pp. 3555–3561. AAAI Press (2017)
10. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp. 854–860 (2013)
11. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: Barjis, J., Pergl, R., Babkin, E. (eds.) EOMAS 2015. LNBIP, vol. 231, pp. 20–36. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24626-0_2
12. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 24:1–24:37 (2015)
13. Donadello, I., Riva, F., Maggi, F.M., Shikhizada, A.: Declare4Py: a python library for declarative process mining. In: BPM Demos. CEUR Workshop Proceedings, vol. 3216, pp. 117–121 (2022)
14. Gunther, C.W., Verbeek, H.: XES-standard definition (2014)
15. Leno, V., Dumas, M., Maggi, F.M.: Correlating activation and target conditions in data-aware declarative process discovery. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 176–193. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_11
16. Maggi, F.M., Di Ciccio, C., Di Francescomarino, C., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Inf. Syst. **74**(Part), 136–152 (2018)
17. Maggi, F.M., Dumas, M., García-Ba nuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: BPM, vol. 8094, pp. 81–96 (2013)
18. Maggi, F.M., Marrella, A., Patrizi, F., Skydanienko, V.: Data-aware declarative process mining with SAT. ACM Trans. Intell. Syst. Technol. (2023). https://doi.org/10.1145/3600106
19. Maggi, F.M., Montali, M., Bhat, U.: Compliance monitoring of multi-perspective declarative process models. In: EDOC, pp. 151–160 (2019)
20. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: BPM, vol. 6896, pp. 132–147 (2011)

21. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), pp. 287–300 (2007)
22. Schunselaar, D.M.M., Maggi, F.M., Sidorova, N.: Patterns for a log-based strengthening of declarative compliance models. In: IFM, vol. 7321, pp. 327–342 (2012)
23. Skydanienko, V., Di Francescomarino, C., Ghidini, C., Maggi, F.M.: A tool for generating event logs from multi-perspective declare models. In: BPM (Dissertation/Demos/Industry). CEUR Workshop Proceedings, vol. 2196, pp. 111–115. CEUR-WS.org (2018)